Second Practical Exercise

Programming

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Programming – Grado en Ingeniería Informática	
Authors *Of the English version:* Juan Gómez Romero *Based on the work by:* Ángel García Olaya Manuel Pereira González Gustavo Fernández-Baillo Cañas	

Second Practical Exercise

This practical exercise has three parts:

- The first part is a tutorial aimed to introduce the programming environment (J2SE 1.6 and Eclipse). **No deliveries** will be made for this part.
- The second part is aimed at introducing Java programming. This part includes a set of basic exercises to be developed by the students, and a set of questions associated to each exercise to be answered. The code of these exercises will not be delivered. Students **have to create a video**[1] (*screencast*) explaining the solutions to 4 of the proposed exercises. These exercises will be assigned by your professor one week before the deadline. In addition, this part will be also marked **by filling an individual test** in Aula Global. This test will be held in the first lab class after the deadline.
- The third part encompasses more complex exercises. The **code of these exercises must be delivered**.

1. First part: Introduction to Eclipse IDE

The first part of the second practical exercise is a tutorial aimed at introducing the tools that are used in this lecture. **No deliveries will be made for this tutorial part**.

1.1 Java SE

The *Java Development Kit* (JDK) is the standard platform for general-purpose Java program development. JDK includes, among others:

a) The *javac* compiler. The compiler **verifies the syntax** of the Java source files (.java files) and **converts** them to compiled ***bytecode*** files (.class files).

b) The *java* virtual machine. The *Java Virtual Machine* (JVM) **interprets** the *bytecode* files (.class) and runs the program.

c) The *javadoc* program. *javadoc* automatically generates program **documentation**.

d) A complete class catalogue with **utilities** for: file management, creation of graphical interfaces, communications, etc.

1.1.1 *javac* compiler

javac is the program that transform Java source files to bytecode files. Bytecode files can be interpreted by the virtual machine. *javac* reads a .java source file and generates a corresponding .class file with bytecodes for each class included in it.

For instance, this command on the system command window:

```
> javac HelloWorld.java
```

Creates a HelloWorld.class file associated to the class defined in HelloWorld.java[2] *file*.

1.1.2 Java Virtual Machine (JVM)

The *JVM* is a program that interprets the bytecode files. The *JVM* is an abstract layer between Java programs and the operating system/hardware in which they are executed. In this manner,

---

[1] Students are free to choose any setup for the video: a close-up of the student explaining the solution to the camera with his/her words, a screencast without voice, etc. Software like *CamStudio* (http://camstudio.org/) can be used to record the screen, if necessary –read carefully the features and the help documents of this tool.

[2] Notice that the class and the .java file that contains the class definition have the same name, including upper and lowercase letters.

Java programs are portable --which means that they can be executed in different platforms without changing the source code--, because they are interpreted by *JVMs* with exactly the same functionalities.

The call to the *JVM* program in the command line has a parameter: the name of the bytecode file created from the source file that has a class with a *main* method. All the *.class* files mentioned by the class with the *main* must be accessible (in a proper folder structure), in order to load them:

> *For instance, this command:*
>
> **`> java HelloWorld`**
>
> *Executes the HelloWorld class previously compiled. To execute a class, it must contain the method* public static void main (String [] args)*. Java will execute all the instructions in this method sequentially.*

1.1.3 javadoc

javadoc is a tool to automatically generate the documentation of the classes contained in a Java source file:

> *For instance, this command:*
>
> **`> javadoc HelloWorld.java`**
>
> *Generates a HTML file named HelloWorld.html with documentation for this class in the format of the Java API (Application Programming Interface).*

1.1.4 JDK class packages

JDK encompasses a complete class repository to facilitate the development of programs that access the file system, create graphical interfaces, interchange data through a communication network, etc. The documentation of the JDK 6 classes can be found at: http://java.sun.com/javase/6/docs/api/.

1.1.5 Program development with the tools of the JDK

Usually, programmers use an integrated development framework that makes it easier to create, test, and run source code. Nevertheless, the utilities of the JDK can be directly used. This exercise explains how to compile and execute Java programs without using an IDE.

This exercise is not to be delivered.

Exercise 1. Direct use of J2SE

a. Create a file named *HelloWorld.java* with the contents described in Appendix I. Use any text editor (e.g. the *notepad* in Windows) and save it in the Desktop. Be sure that the file has .java extension, and not .java.txt.
b. Open a system console. In Windows 7, click Start and type "cmd" in the search box. An alternative is to click on Start >> All programs >> Accessories >> Command Prompt.
c. In the command line, type "cd Desktop".
d. Compile the Java file to transform it into bytecodes with `javac`. Be sure that a .class file has been created[3].
e. Interpret the bytecodes with the `java` program.

[3] If Windows does not find the `javac` program, before running `javac` the following command must be executed: `path = "path of the javac file"`.

1.2 Eclipse

Eclipse is an *open source*[4] Java-based graphical IDE (integrated development environment). The Eclipse IDE is organized in *perspectives*. Each *perspective* shows the IDE functionalities useful in a concrete task. For instance, the default *Java perspective* is intended for writing Java source code and executing Java programs, whereas the *Debug perspective* is intended for code debugging. The current Eclipse perspective can be changed by selecting *Window* >> *Open perspective* in the menu, or by clicking the button on the top-right corner of the screen. As mentioned, we will use the Java perspective for Java programming.

1.2.1 Running Eclipse

The first time Eclipse in launched (and successively, if desired), the user is asked for the *workspace* location. The workspace is the directory where Eclipse will save the programs created by the user (.java files) and the corresponding compiled bytecodes (.class files), as well as the information related to the software projects. Students are recommended to use a pendrive and choose as the workspace a directory of the USB file system. To do this, a directory must be created in the pendrive (named *Java*, for instance) and designated as workspace when launching Eclipse.

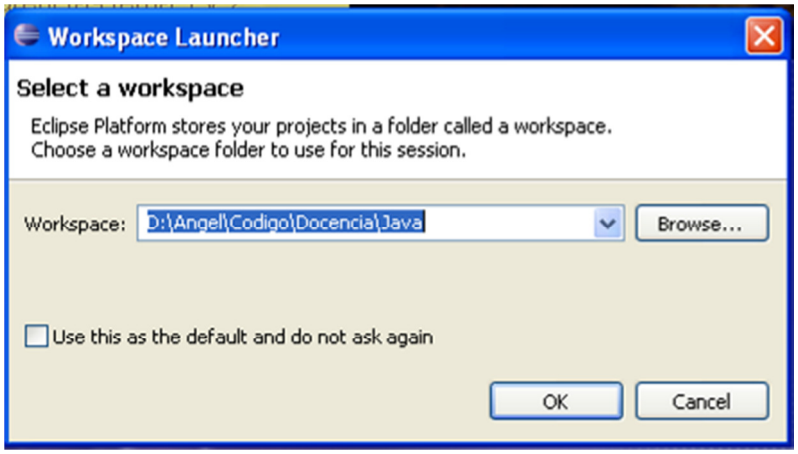

Figure 1: Selection of the workspace

1.2.2 Project creation

Eclipse manages Java developments with projects; that means that in order to create a Java program in Eclipse, it is necessary to create a project before. To create a project, the following steps must be completed:

1. Select *File* >> *New* >> *Java Project* in the menu.
2. Fill in the project data. For simple projects --like the ones that we will create in this lecture--, only the project name needs to be specified. Create a project named *Practice*, which will be used in this exercise and in the next one.
3. Once project data has been typed, click on the *Finish* button.

1.2.3 Package creation

A package is a folder inside the Eclipse project used to store Java source files. By default, Eclipse saves class source files in a default package. It is recommended to create at least one package in a project. To create a package:

1. *File* >> *New* >> *Package* in the menu.

[4] Open source means that the source code of the program is available and, under certain conditions, it can be changed and distributed. More information on open source software: http://www.opensource.org/docs/definition.php.

2. Assign a name for the package in the package creation dialog (`practicalex2`) (The convention is to name packages starting with a lowercase letter).

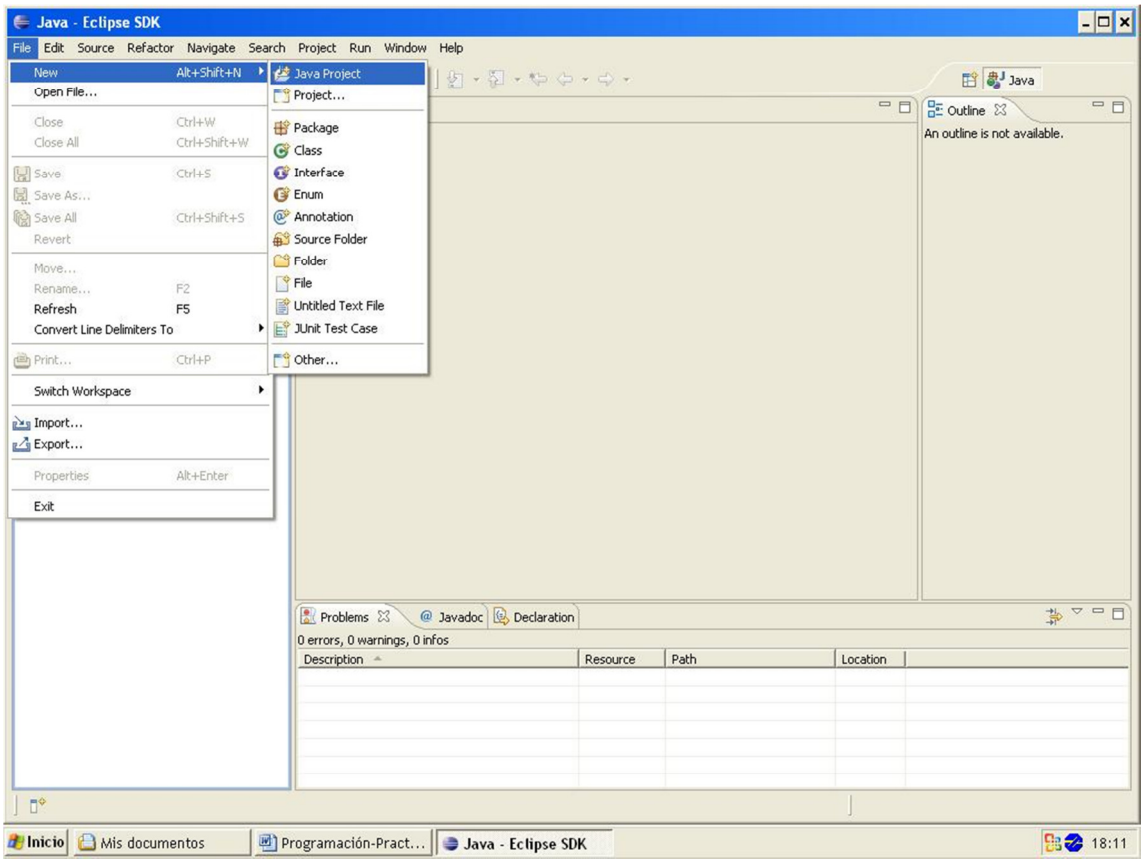

Figure 2: Project creation

As a result, Eclipse creates a folder for the project in the workspace with two files: *.project* and *.classpath*. These files do not contain Java code, because they are internally used by the IDE. The Java source code is saved in the *src* sub-folder. The compiled Java code is saved in the *bin* subfolder. The active workspace can be changed by selection *File >> Switch Workspace* in the menu.

1.2.4 Class creation

To create a Java class inside a project, the following steps must be carried out:

1. Select *File>> New >> Class* in the menu.
2. Select the package of the class (`practicalex2`).
3. Type a name for the class in the class creation dialog which appears next (e.g.: *HelloWorld*). The convention is to name Java classes starting with an uppercase letter.
4. Specify if a *main* method must be created for this class by ticking the corresponding check box. (This is not necessary in our example, because we will change all the contents of the file with the code in Appendix I).
5. Click the *Finish* button.

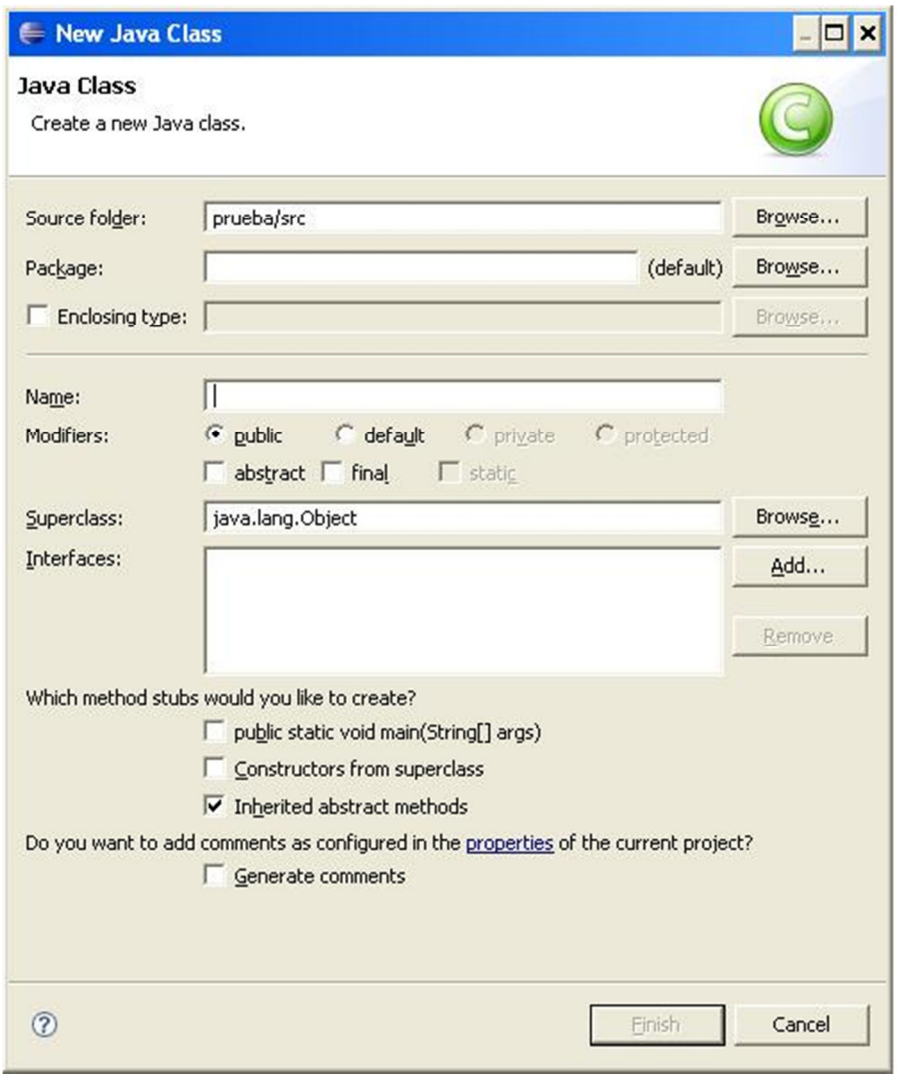

Figure 3: Class creation dialog

Eclipse creates a file named <name of the class>.java with the source code corresponding to the class skeleton. Next, the code implementing the class must be typed (inside the `main` class). As mentioned, we will change all the contents of the file but the first line --the package of the class.

1.2.5 Program running

Programs can be executed in the *Java* perspective by clicking on *Run* >> *Run* in the menu, or alternatively, by typing *Control* + *F11* in the keyboard. Running the program makes Eclipse show the result of the execution on the console.

1.2.6 Program running with arguments[5]

As it occurs when invoking a Java program from the command-line, Eclipse allows changing the arguments (also named program parameters, or simply parameters) passed to the *main* method of the class. To change these values, select *Run* >> *Open Run Dialog* in the menu. In the *Arguments* tab, we can specify a suitable list of arguments, one per line. Each argument will be

[5] This section is recommended to be read when solving the exercise 28 of the second part of the practical exercise.

stored in successive positions of the *args* array, i.e. the first one is *args[0],* the second one is *args[1]*, etc.

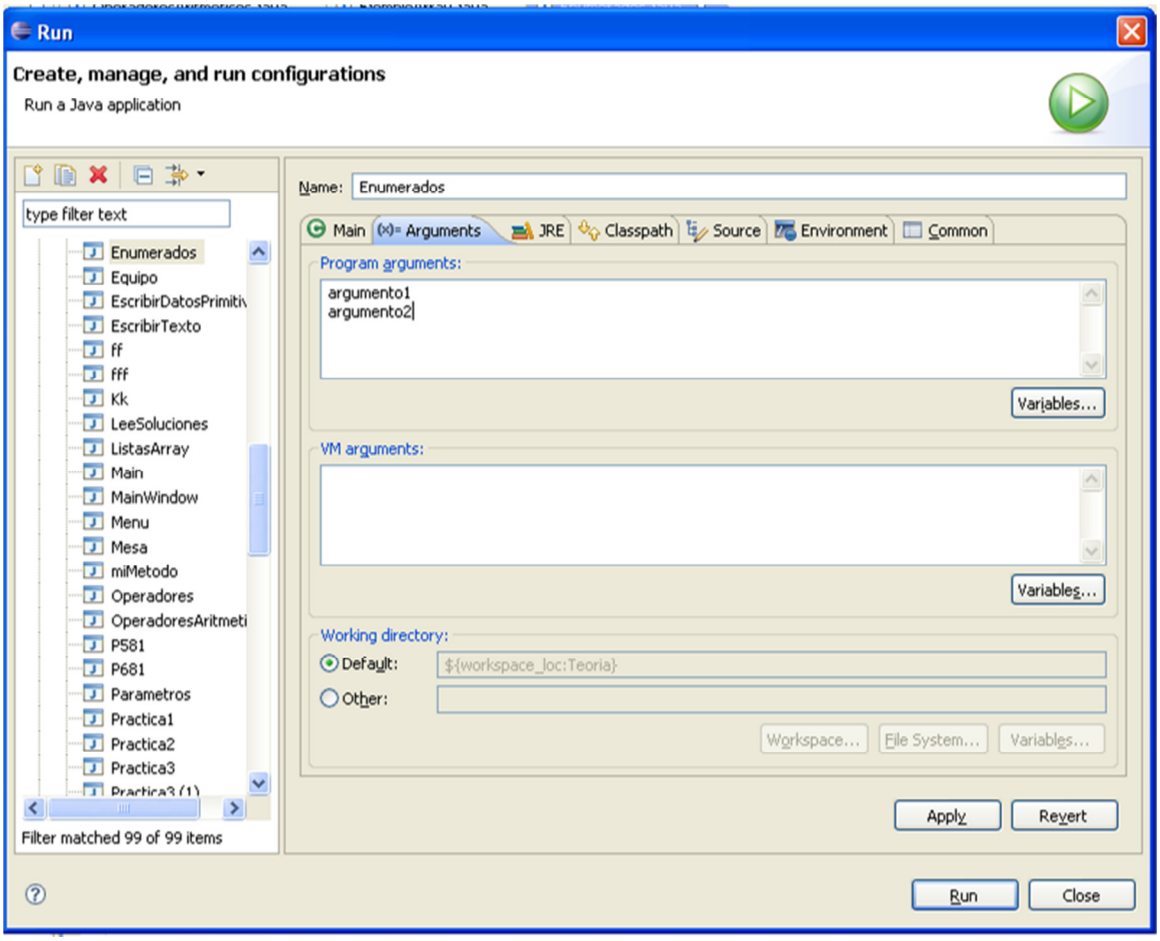

Figure 4: Arguments in the execution of a Java program

1.2.7 Program debugging

The Eclipse *debug* perspective is intended for detecting and correcting program runtime errors, which is known as debugging. The *debug* perspective can be activated by selecting *Window* >> *Open perspective* >> *Debug* in the menu. The execution of programs with the *debug* perspective can be analyzed by:

1. Pausing the execution of the program when it reaches a break point. The menu option *Run* >> *Toggle BreakPoint* allows for specifying a line of the code where the execution of the program is paused. After defining a break point, the program can be launched in debugging mode by selecting *Run* >> *Debug*. The execution of the program will be paused in the first break point.
2. Executing the program step by step (pausing the execution after each instruction). The menu options *Run* >> *Step into* and *Run* >> *Step Over* allows for executing the program line by line.

The debug perspective also allows programmers to check the value of variables during the execution of the program in debug mode. The *Variables* tab list the variables which are active in the current scope and their values. It is also possible to add new *watches* to check the value of other variables or expressions. To do so, we have to select the expression to watch in the code editor, right click and select *Watches*. The expression and its value will be shown on the *Watches* tab. Watched expressions can be edited and removed.

Using the debugger is strongly recommended to detect and fix runtime error. Using *println* instruction usually leads to new errors and difficulties to find program bugs.

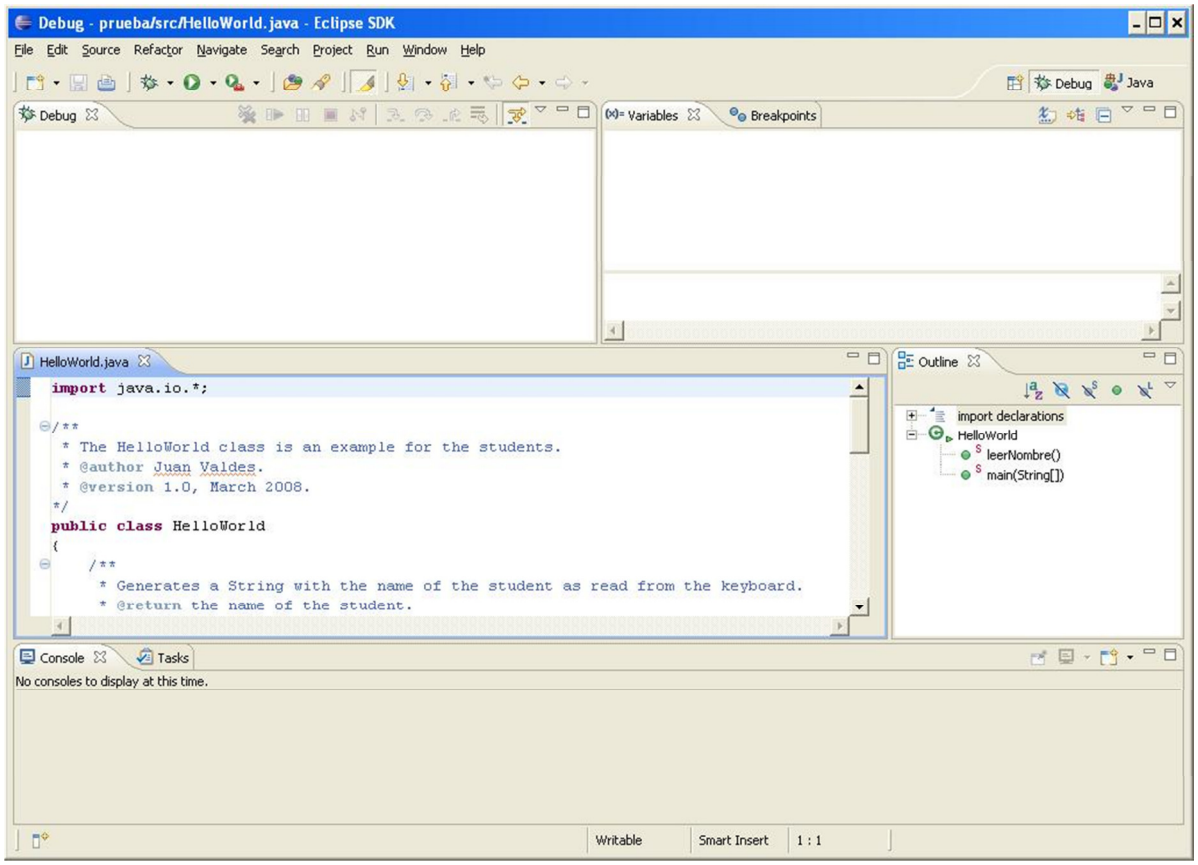

Figure 5: Debug *perspective in Eclipse*

Exercise 2. Introduction to Eclipse

a. Create a Java Project including the *HelloWorld* class described in Appendix I.
b. Execute the main method of the *HelloWorld* class.
c. Run the main method of the *HelloWorld* class step by step (Debug mode).
d. Watch the value of the *name* variable.

2. Second part: Introduction to Java programming

The second part of the second practical exercise is aimed at applying the knowledge learnt in lectures 2-4. In that regard, various exercises of increasing difficulty are proposed. Solutions to this second part must be delivered for marking –that is, no code must be delivered and a video must be created. This part is also evaluated with a test in Aula Global.

2.1 Basic Java data types, arrays and operators[6]

Note: Do not deliver the code of the exercises in this section. Instead, students must answer the questions in their reports and complete the test in Aula Global (date to be scheduled). The exercises should be developed with the Eclipse environment.

Exercise 1. Variable declaration

Create a class named `Exercise1`. Inside this class, create a `main` method and, inside this method, declare eight variables, each one with a different basic Java data type (one declaration per sentence). Declare a `String` variable. Next, assign valid values to each variable. Print variable values on the screen with `System.out.println(<variable name>)` instructions, replacing `<variable name>` by the name of each variable.

Exercise 2. Basic data types range

Continue the previous exercise (do not change anything, just append the new instructions below the last `System.out.println` instruction) by adding assignment instructions for the numeric variables with values out of the ranges of each one of the types. Print the results on the screen. What happens in each case?

Exercise 3. Numeric types precision

Continue the previous exercise (do not change anything, just append the new instructions below the last `System.out.println` instruction) by declaring two `float` variables. Initialize the first variable with the value 1234567890F and the second one with the value 1234567899F. What is the result? Repeat the same procedure with two integer variables. What is the result?

Exercise 4. Variable use

Create a class named `Exercise4`. Inside this class, create a `main` method and, inside this method, declare a variable (of any type) without an initial value. Print this variable on the screen before any value assignment. What happens? Why?

Create a second variable of the same type and make it equal to the first variable. What happens? Why?

Exercise 5. Multiple declarations

Create a class named `Exercise5`. Inside this class, create a `main` method and, inside this method, declare in a single line 3 variables of the same basic type (anyone of them).

Declare in a single sentence three variables of a type different from the previous one, assigning a value to the first one and to the last one in the same declaration. Next, in other sentence, assign a value to the second variable. Declare a variable of the same type and assign a value resulting from the combination of them (e.g., the sum of all of them). Print the four variables on the screen.

---

[6] Some of the following exercises have been designed with errors. In these cases, the report written by the student must explain which these errors are and why they occur. In order to continue the execution of the incorrect program, once the errors have been located and explained, please add two slashes (//) before the line that produces the error. This way, Java will consider this line as a comment and the error will be eluded.

Add below an instruction that changes the value of the first variable of this second set. Print the value of the forth variable. Does the forth variable value (the 'combination' variable) change after modifying the first variable? Why?

Exercise 6. Constants

Continue the previous exercise by adding `final` before de declaration of the second set of variables. What happens? Why?

Remove the previous `final` modifier and add `final` before the declaration of the forth variable. What happens? Why?

Create a `String` constant and provide a value. In the next line, change its value. Is it possible?

Exercise 7. Text strings (`char` data type)

Create a class named `Exercise7`. Inside, create a `main` method and, inside this method, type the following code:

```
char a;
a = '\\\';
System.out.println(a);
```

What is the result of the execution of the previous sentences? Why?

Is it possible to create a `char` with more than a character (for instance, `char b = 'hola'`)?

Exercise 8. Variable copy

Create two variables of anyone of the basic types, assign a value to the first one, and make the second one equal to the first one. Print on the screen the second variable. Extend the program with a new instruction that changes the value of the first variable, and add other instruction to print again the second one. Does the second variable change its value? Why? Repeat the same procedure for two `String` variables.

Exercise 9. Array declaration (I)

Create a class named `Exercise9`. Inside, create a `main` method. Inside the `main`, declare an array of any integer data type, an array of any real data type, a `char` array, a `boolean` array, and a `String` array. Create these arrays with different sizes. In the next sentences, assign values to some of the elements of the arrays. What happens when we use (e.g. print) some of the elements that have not been initialized? What is the difference with respect to exercise 3?

Create an array of any type providing an initial value for its elements in the declaration.

Use `length` to print on the screen the size of the previous arrays. Change the length of one of these arrays by typing `array.length = 5`. What happens? Why?

Select one of the previous arrays and make equal two array elements (e.g. a[5] = a[3]). Print both elements. Next, add a line changing the value of the second element, and print both elements again. Does the first one also change? Why?

Exercise 10. Array copy

Create a class named `Exercise10`. Inside, create a `main` method. Create an array of any type. Provide a value for all its elements. Create a second array and make it equal to the first one (with `a = b`). Change the value of an element of the first array. Does the corresponding element of the second array change? Why?

Create two additional arrays and repeat the previous steps, but instead of `=` use `System.arraycopy(source, first element, destination, first element, number of elements)`. Can you notice any difference?

Exercise 11. Multiple dimension arrays

Create a class named `Exercise11`. Inside, create a `main` method. Create a two dimension array of any type and assign values to its elements in the declaration.

Create a two dimension array of a different type. Assign values to its elements one by one.

Exercise 12. Length of the dimensions of an array

Create a class named `Exercise12` and its corresponding `main` method. Define a 3x4 `String` array and assign values to its elements. Print on the screen the number of rows and columns of the array by using the `length` property.

Exercise 13. Arithmetic operations with integers (I)
Create a class named `Exercise13` and its corresponding `main` method. Type the following code inside:

```
int a;
int b;
int c;
int d;
a = 2;
b = 3 * 3;
c = 7 / 3;
d = a + b * c;
System.out.println(d);
```

What is the result of the previous code? Why?

Exercise 14. Arithmetic operations with integers (II)
Create a class named `Exercise14` and its corresponding `main` method. Declare three variables for each one of the four integer-based (`byte`, `short`, `int`, `long`) data types (12 variables) and three `char` variables. For each one of the types (integers and chars), provide initial values for the first two variables, and assign to the third one the result of operations involving all the Java arithmetic operators (adding, subtraction, etc.). Print the results on the screen.

Do you get any error? Why? (Note: Once discovered the errors, add // in the beginning of the wrong lines to convert them in comments, instead of erasing them).

Exercise 15. Arithmetic operations with integers (III)
Create a class named `Exercise15` and its corresponding `main` method. Declare three `int` variables. Assign value 5 to the first one, and value 0 to the second one. Assign the third variable the result of dividing the first variable by the second one. Print the result on the screen. Is there any error? Why? Does the result change if the variables are declared with other integer type instead of `int`?

Exercise 16. Autoincrement
Create a class named `Exercise16` and its corresponding `main`. Declare two `long` variables, assign a value for the first one, and make the second one equal to the autoincrement of the first one (with the prefix autoincrement operator). Print them.

Next, create two additional variables and repeat the same code, but using the postfix autoincrement operator. Print them. Is there any difference? Why?

Repeat these steps for the autodecrement operator.

Exercise 17. Arithmetic operations with char
Create a class named `Exercise17` and its corresponding `main` method. Declare a `char` variable with a suitable initial value. Print it with `System.out.println(variable+1)`. What happens? Why? Print it with `System.out.println(++variable)`. What happens? Why?

Exercise 18. String concatenation
Create a class named `Exercise18` and its corresponding `main` method. Declare three `String` variables. Assign any value to the first two variables and make the third one equal to the first one + second one. Print the third variable. What happens? What happens if you make third = first – second?

Exercise 19. Arithmetic operations with real numbers
Repeat exercises 13, 14 and 15, but use real types (`double` or `float`) instead of `int`. Can you see any difference?

Exercise 20. Automatic type conversion
Create a class named `Exercise20` and its corresponding `main`. Declare and assign an initial value to variables of each basic data type, including `String`. Next, assign one variable to the other ones, and repeat for each variable. In your report, complete the next table with the valid assignments. Write "YES" if the variable in the row can be assigned to the variable in the column (i.e. <column variable> = <row variable>; is possible); otherwise write "NO".

Type	byte	short	int	Long	float	double	char	boolean	String
byte	**YES**								
short		**YES**							
int			**YES**						
long				**YES**					
float					**YES**				
double						**YES**			
char							**YES**		
boolean								**YES**	
String									**YES**

Exercise 21. User-forced type conversion
Create a class named `Exercise21` with the same contents as `Exercise20`. Use the casting operator to force the conversion between all types, and fill in the table. Are there remaining "NO" cells? If some cell is still "NO", answer why.

Exercise 22. Type conversion
Create a class named `Exercise22` with a corresponding `main` method. Type the following code inside the `main` method:

```
char a;
int b;
short c;
long d;
float e;
double f;

System.out.println("we assign a char var. to integers");
a = '4';
b = a;
c = a;
d = a;
System.out.println("a:" + a);
System.out.println("b:" + b);
System.out.println("c:" + c);
System.out.println("d:" + d);

System.out.println("we assign a double var. to a float");
f = 1e200;
e = f;
System.out.println("f:" + f);
System.out.println("e:" + e);
```

```
System.out.println("we assign a float var. to an int");
e = 1234.5678;
b = e;
System.out.println("e:" + e);
System.out.println("b:" + b);
```

What is the result of the previous program? Why?

Exercise 23. Relational operators

Create a class named `Exercise23` with a corresponding `main` method. Type the following code inside the `main` method:

```
int a,b;
float c=3;
boolean r,s,t,u,v,w,x;
a = 3;
b = 8;
r = a == 0;
s = a != 0;
t = a <= b;
u = b >= a;
v = b > a;
w = b < a;
x = c == 3.0;
System.out.println("r:" + r);
System.out.println("s:" + s);
System.out.println("t:" + t);
System.out.println("u:" + u);
System.out.println("v:" + v);
System.out.println("w:" + w);
System.out.println("x:" + x);
```

What is the result of the previous program? Why?

Exercise 24. Debugging programs with basic data types

Run the program developed in the previous exercise in debug mode –use a breakpoint and step by step execution. Can you see how the values of the variables change? Which are the advantages of the debugger in this case?

Exercise 25. Logic operators

Create a class named `Exercise25` with a corresponding `main` method. Type the following code inside the `main` method:

```
int a,b;
boolean r,s,t;
a = 3;
b = 8;
r = a == 0 | b >= a;
s = a != 0 & b < a;
t = a <= b ^ b > a;
System.out.println("r:" + r);
System.out.println("s:" + s);
System.out.println("t:" + t);
```

What is the result of the previous program? Why?

Exercise 26. Assignment operators

Create a class named `Exercise26` with a corresponding `main` method. Type the following code inside the `main` method:

```
int a=5,b=3;
boolean r=true,s=false;
a+=b+8*b;
r&=s;
System.out.println("a:" + a);
System.out.println("b:" + b);
System.out.println("r:" + r);
System.out.println("s:" + s);
```

What is the result of the previous program? Why?

Exercise 27. Operator precedence

Create a class named `Exercise27` with a corresponding `main` method. Type the following code inside the `main` method:

```
int a=5,b=3,c=20,d=20;
c-=++a/b-3+a%b;
d-=++a/(b+3-4*a)%b;
System.out.println("c:" + c);
System.out.println("d:" + d);
```

What is the result of the previous program? Why?

Exercise 28. Variable scope

Create a class named `Ejercicio28` with a corresponding `main` method. Inside the main method, declare a variable of any type. Next, declare again the same variable (same name and same type). Is it possible? Why? What happens if the name is the name, but the type is different?

Enclose the declaration of the first variable between two braces `{}`. Can you declare now the same variable out of the braces? Why?

Assign a value to the variable inside the braces. Print it out of the braces. What happens? Why?

Exercise 29. Irregular arrays

Create a class named `Exercise29` with a corresponding `main` method. Define a two-dimension irregular array of String named `year`. Each row will represent a month, and the columns of each row will represent the days of the month (in a non-leap year). That is, the first row should have 31 elements, the second one 28, etc. Assign the value "no class today" to the cell corresponding to January, 8th. Assign other values to the cell corresponding to today. Print on the screen the number of rows and columns of the array by using the `length` property.

Exercise 30. Arguments of the `main` method

When executing the main method of a class, we can add execution parameters (see the Eclipse tutorial in Part I). These parameters are stored in an array of `String` named `args`. This array, specified in the declaration of the main, is automatically created and assigned by Java.

Create a class named `Exercise30` with a corresponding `main` method that prints on the screen the first three arguments passed to the program. What happens if the program is called with less than three arguments? And is it is called with more than three arguments?

Exercise 31. Loops and arrays

Define an array of 100 integer elements. Use a `while` loop to initialize the array with the numbers from 100 to 199. Next, use a `while` loop to print on the screen the contents of the array.

Exercise 32. Objects as data structures

Define a class named `Student` with three attributes: `name` (String), `surname` (String), `average_mark` (double). Create a second class named `Exercise32` with a `main` method. Declare a `Student` object reference and print directly this variable on the screen. What happens? Initialize the object reference with a `null` value. Print the variable and the `name` attribute. What happens? Why? Add the proper code to initialize the object reference –use the `new` operator. Print the variable and the `name` attribute. What happens?

Exercise 33. Attribute initialization

Extend the previous exercise to print all the object attributes on the screen. What happens? Assign values to the object attributes in the `main` method and print them again. What happens? Modify the implementation of the `Student` class to assign default values to the attributes: `name` and `surname` must be "??", `average_mark` must be 0. Declare a second `Student` object reference and initialize it with the `new` operator. Print the attribute values of this second object on the screen. What happens? Which is difference with respect to the previous case? Assign other values to the attributes of the second object. Is it possible?

Exercise 34. Direct object assignment

Extend the previous exercise by assigning the first `Student` object to the second `Student` object. Print on the screen the values of the attributes of both objects. Assign the value "John" to the `name` attribute of the second object. Print on the screen the value of the `name` attribute of the first object. What happens?

Exercise 35. Debugging programs with objects

Run the program developed in the exercises 33 and 34 in debug mode –use a breakpoint and step by step execution. Can you see how the values of the objects and their attributes change? Which are the advantages of the debugger in this case?

3. Third part: Advanced exercises

Exercise 1. Division

Create a program that reads two integer values from the keyboard and stores them in two integer variables, respectively `value_a` and `value_b`. If `value_a` is divisible by `value_b`, the program must print "[a] is divisible by [b]" ([a] and [b] must be the actual values of the variables `value_a` and `value_b`). If `value_a` is not divisible by `value_b`, the program must print "[a] is not divisible by [b], the remainder is [r]" (being [r] the actual value of the reminder of a/b).

Exercise 2. Rock-paper-scissors

Create a program to calculate who is the winner of a "rock-paper-scissors" game. The program must read two `String` values from the keyboard each one representing the selection of a player (`selection_1`, `selection_2`). The program must print on the screen which player is the winner of the game and the players' gestures. For example, if `selection_1` is "Rock" and `selection_2` is "Paper", the program must print "Player 2 wins. Paper defeats Rock". If both players make the same gesture, the program must print "Tie game!"

Select in Eclipse the option "Show line numbers" (*Preferences >> Text editors --> Show line numbers*). Run the program step by step with the debugger, using "Paper" and "Rock" as the inputs, and write down the sequence of line numbers resulting from this execution. Include this sequence as a comment at the end of the program.

Exercise 3. Football pool

Create a program that declares a 2-dimension `char` array of15 rows x 3 columns. Use nested loops to set a '1' to the elements in the first column, 'X' to the elements in the second column, and '2' to the elements in the last column. Use nested loops to print the array contents on the screen. Change the value of any cell in the array (set '?' value) and print it again.

Exercise 4. Exponentiation

Create a program to print the powers of an integer number read from the keyboard. For example, if the read value is 3, the program must print the output below. NOTE: use the *Math.pow(base, exponent)* method to make the calculus.

3^0 = 1

3^1 = 3

3^2 = 9

3^3 = 27

…

3^10 = 59049

Exercise 5. Daytime

Create a program to calculate the day period corresponding to a given hour read from the keyboard. The program must check if the received hour is between 0 and 23, and print (by using a `switch` instruction):

"`morning`" when the hour is between 6 and 12

"`afternoon`" when the hour is between 13 and 21

"`evening`" when the hour is between 22 and 2

"`night`" when the hour is between 3 and 5

Exercise 6. Draughts

Create a program that defines an array to represent a draughts 10x10 board with the initial position of all the pieces. Print the board on the screen. Implement the first movement of one of the white pieces situated in one of the most advanced position. The piece to be moved must be read from the keyboard –an integer value in {1, …, 5}. The destination position must be also

read from the keyboard –an integer value in {1, 2}; being 1 the position on the left, and 2 the position on the right. Print the board resulting from the move. If the combination of the piece and the position is not valid, print "Not valid move!".

The board must be represented as follows:

- Empty squares: '@' symbol for empty light squares; '#' symbol for empty dark squares.
- Non-empty squares: 'd' for squares with a white piece on; 'D' for squares with a black piece on.

Source: http://en.wikipedia.org/wiki/Draughts

Exercise 7. Factorization

Create a program with a `main` method that, given an integer value received as a parameter, prints out the result of its prime factorization.

Exercise 8. Previous second

Create a program with a `main` method that, given a time value, prints out the time corresponding to the previous second. The time values must be passed as arguments to the main method (the first one is the hour, the second one the minutes, the third one the seconds).

Exercise 9. Set inclusion

Create a program with a main method that receives several numerical values as arguments. The values must be stored in 2 arrays named `array1` and `array2`; the first half of them in `array1`, the remaining ones in `array2`. If the number of values is odd, the last value is discarded. The program must check if all the values in `array2` are included in `array1`. If this condition holds, the program must print "set 2 is included in set 1"; otherwise, the program must print "set 2 is not included in set 1".

NOTE: Repeated values are allowed in both arrays.

Exercise 10. Blackjack

Create a program to play a simplified version of the blackjack game for one player.

The game consists of several iterations in which cards are drawn from the deck managed by the computer. Each card as a numerical value; the value of the drawn card is added to the player score. Accordingly, at each iteration, the program must generate a random card. Cards will be objects of the `Card` class, which must have three attributes: value (`int`), name (`String`), and suit (`String`: clubs ♣, diamonds ♦, hearts ♥ and spades ♠) (see NOTE). The value of a card is calculated as follows: aces 1 point, figures 10 points, remaining cards have their numerical value.

After updating the player score, the program must print the name of the card, its value and the player score. The player selects whether he/she wants to continue or not. The player wins either if he/she: (a) obtains 21 points; (b) obtains less than 21 points, but the score is closer to 21 than the score of the computer –the points of the computer are calculated as a random number in {1, …, 21}. The player loses if he/she: (a) obtains more than 21 points; (b) obtains less than 21 points, but this number is not closer to 21 than the score of the computer.

NOTE: It is not necessary to check if a card has been previously generated; i.e., repeated cards are allowed.

Appendix I: HelloWorld.java

```
/**
 * The HelloWorld class is an example for the students.
 * @author Juan Valdes.
 * @version 1.0, March 2008.
*/

import java.util.Scanner;

public class HelloWorld {
	/**
	 * Generates a String with the name of the student as read
	 * from the keyboard.
	 * @return the name of the student.
	 */
	 public static String readName(){
		Scanner sc = new Scanner(System.in);

		System.out.println("Enter your name and type Enter: ");
		String name = sc.nextLine();

		return name;
	}

	/**
	 * Says Hello to the student.
	 * @param args the set of command-line arguments.
	 * @return .
	 * @exception .
	 */
	public static void main(String[] args) {

		if(args.length > 0){
			System.out.println("Hello " + args[0]);
		} else{
			System.out.println("Hello "+ readName());
		}
	}
}
```